Correspondence between operational and declarative concurrency semantics

Ori Lahav Viktor Vafeiadis

29 August 2017

Definition (Operational SC)

An outcome *O* is allowed for a program *P* under SC if there exists *M* such that $P, S_0, M_0 \Rightarrow^* \mathbf{skip} \parallel ... \parallel \mathbf{skip}, O, M$.

Definition (Declarative SC)

An outcome O is allowed for a program P under SC if there exists an SC-consistent execution graph of P with outcome O.

> How do we show that the two definitions are equivalent?

Operational version of SC declarative semantics

State:

 $\langle \textit{P},\textit{S},\textit{G},\texttt{sc}\rangle \in \mathsf{Program} \times (\mathsf{Tid} \rightarrow \mathsf{Store}) \times \mathsf{ExecutionGraph} \times \mathcal{P}(\mathsf{Event} \times \mathsf{Event})$

- Initial stores: $S_0 \stackrel{\triangle}{=} \lambda i. s_0$
- Initial execution: G_0 consisting only of the initialization events
- ▶ Initial sc-relation: sc_0 is an arbitrary total order on G_0 .E

$$\begin{array}{l} \text{NON-SILENT} \\ P, S \xrightarrow{\text{tid}(a): \text{lab}(a)} P', S' \\ \hline P, S, G, \text{sc} \Rightarrow P', S', G, \text{sc} \end{array} \end{array} \xrightarrow{\text{NON-SILENT}} P, S \xrightarrow{\text{tid}(a): \text{lab}(a)} P', S' \\ G' \in \text{Add}(G, a) \\ \text{sc}' = \text{sc} \cup (G.\text{E} \times \{a\}) \\ \hline G' \text{ is SC-consistent wrt sc}' \\ \hline P, S, G, \text{sc} \Rightarrow P', S', G, \text{sc} \end{array}$$

where Add(G, a) is the set of all complete graphs G' satisfying:

Operational version of SC declarative semantics

Definition (Operational-declarative SC)

An outcome *O* is allowed for a program *P* under SC if there exist *G*, sc such that $P, S_0, G_0, sc_0 \Rightarrow^* skip || ... ||skip, O, G, sc.$

Establish correspondence between operational SC and declarative SC in two steps:

- 1. operational SC = intermediate SC
- 2. declarative SC = intermediate SC

Operational SC = intermediate SC

We will use forward weak *simulation*. Consider two labeled state transition systems $M_1 = \langle Q_1, q_1^0, \rightarrow_1 \rangle$ and $M_2 = \langle Q_2, q_2^0, \rightarrow_2 \rangle$.

- $\mathcal{R} \subseteq Q_1 \times Q_2$ is a *simulation relation* from M_1 to M_2 if:
 - $q_1^0 \mathcal{R} q_2^0$, and
 - ▶ whenever $q_1 \mathcal{R} q_2$ and $q_1 \rightarrow_1 q'_1$, then there exists some $q'_2 \in Q_2$ such that $q_2 \rightarrow^*_2 q'_2$ and $q'_1 \mathcal{R} q'_2$.
- $\mathcal{R} \subseteq Q_1 \times Q_2$ is called a *bisimulation relation* if it is a simulation relation from M_1 to M_2 and \mathcal{R}^{-1} is a simulation relation from M_2 to M_1 .

Lemma

If a simulation relation exists then for every state $q_1 \in Q_1$ that is reachable from q_1^0 in M_1 , there exists some $q_2 \in Q_2$ that is reachable from q_2^0 in M_2 and satisfies $q_1 \mathcal{R} q_2$.

Operational SC = intermediate SC

Our bisimulation relation:

 $\langle P, S, M \rangle \sim \langle P', S', G, sc \rangle$ if the following hold:

- ► *P* = *P*′
- ► *S* = *S*′

•
$$M = \lambda x$$
. val_w(max_{sc} $G.W_x$)

► *G* is complete and SC-consistent wrt sc.

- Show that ~ is a bisimulation relation.
- Deduce that operational SC and intermediate SC have the same outcomes for any given program.

$\mathsf{Declarative}\ \mathsf{SC} = \mathsf{intermediate}\ \mathsf{SC}$

Two directions:

- \subseteq Every outcome allowed for *P* according to declarative SC is allowed according to intermediate SC
- ⊇ Every outcome allowed for P according to intermediate SC is allowed according to declarative SC

Reminders:

Definition

G is an execution graph of a program *P* with an outcome *O* if G^i is an execution of P(i) with final store O(i) for every $i \in \text{Tid}$.

Definition (Declarative SC)

An outcome O is allowed for a program P under SC if there exists an SC-consistent execution graph of P with outcome O.

Declarative SC \subseteq intermediate SC

Lemma (Execution generation)

Let G be an execution of a program P_0 with outcome O. Let a_1, \ldots, a_n be an enumeration of G.E \ E₀ that respects G.po. Then, there exist $\langle P_1, S_1 \rangle, \ldots, \langle P_n, S_n \rangle$ such that:

•
$$P_n = skip \parallel ... \parallel skip$$
 and $S_n = O$

• For every $1 \le j \le n$, we have:

$$P_{j-1}, S_{j-1} \xrightarrow{\texttt{tid}(a_j):\varepsilon} * \xrightarrow{\texttt{tid}(a_j):\texttt{lab}(a_j)} \xrightarrow{\texttt{tid}(a_j):\varepsilon} * P_j, S_j$$

Declarative SC \subseteq intermediate SC

- Let G be an SC-consistent execution graph of P_0 with outcome O.
- ▶ Let sc be a total order on *G*.E such that *G* is SC-consistent wrt sc.
- ▶ We show that $P, S_0, G_0, sc_0 \Rightarrow^* skip || ... || skip, O, G, sc.$
- Let a_1, \ldots, a_n be an enumeration of $G.E \setminus E_0$ following sc.
- Since $G.po \subseteq sc$, by the previous lemma, there exist $\langle P_1, S_1 \rangle, \dots, \langle P_n, S_n \rangle$ such that:
 - $P_n = \operatorname{skip} \| \dots \| \operatorname{skip} \text{ and } S_n = O$
 - For every $1 \le j \le n$, we have:

$$P_{j-1}, S_{j-1} \xrightarrow{\texttt{tid}(a_j):\varepsilon}^* \xrightarrow{\texttt{tid}(a_j):\texttt{lab}(a_j)} \xrightarrow{\texttt{tid}(a_j):\varepsilon}^* P_j, S_j$$

• For every $0 \le j \le n$, let

- G_j the restriction of G to $E_0 \cup \{a_1, \dots, a_j\}$
- sc_j the restriction of sc to $E_0 \cup \{a_1, ..., a_j\}$

• Then, for every $1 \le j \le n$, we have:

$$P_{j-1}, S_{j-1}, G_{j-1}, \operatorname{sc}_{j-1} \Rightarrow^* P_j, S_j, G_j, \operatorname{sc}_j$$

Operational-declarative SC \subseteq declarative SC

- ▶ Suppose that $P_0, S_0, G_0, sc_0 \Rightarrow^* skip || ... || skip, O, G, sc.$
- ▶ By definition, G is SC-consistent. It remains to show that each Gⁱ is an execution of P₀(i) with final store O(i).
- ▶ We know: $P_0, S_0, G_0, sc_0 \Rightarrow P_1, S_1, G_1, sc_1 \Rightarrow ... \Rightarrow P_n, S_n, G_n, sc_n$ where $P_n, S_n, G_n, sc_n = \text{skip} \| ... \| \text{skip}, O, G, sc.$
- The sequence above induces the following sequence of transitions:

$$P_0, S_0 \xrightarrow{i_1:I_1} P_1, S_1 \xrightarrow{i_2:I_2} P_2, S_2 \xrightarrow{i_3:I_3} \dots \xrightarrow{i_n:I_n} P_n, S_n$$

▶ In turn, by filtering only the transitions of thread *i* we obtain:

$$P_0(i), S_0(i) \xrightarrow{l_{k_1}} P_{k_1}(i), S_{k_1}(i) \xrightarrow{l_{k_2}} \dots \xrightarrow{l_{k_{n_i}}} P_{k_{n_i}}(i), S_{k_{n_i}}(i) = \mathsf{skip}, O(i)$$

▶ It follows that $P_0(i), s_0, G_{\emptyset} \Rightarrow^* \text{skip}, O(i), G^i$, and so G^i is an execution of $P_0(i)$ with final store O(i).

Operational semantics for COH

Recall the following litmus tests:

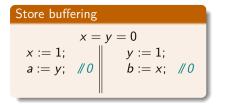
Store buffering
 Constraints

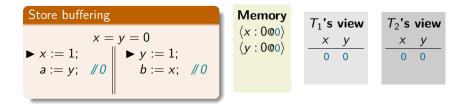
$$x = y = 0$$
 $x := 1$
 $a := y \ \# 0$
 $b := x \ \# 0$

Coherence test	
x = 0	
$\begin{array}{c} x := 1 \\ a := x \not / 2 \end{array}$	x := 2
a := x // 2	b := x // 1

Two approaches:

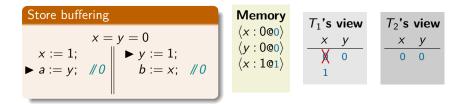
- Out-of-order execution with SC memory.
- In-order execution with non-standard memory:
 - Allow threads to observe different subsets of writes.
 - Use timestamps to order writes to the same location.





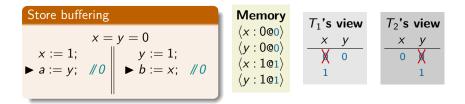
Global memory is a pool of messages of the form

 $\langle location : value @ timestamp \rangle$



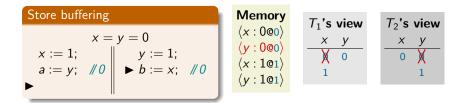
Global memory is a pool of messages of the form

 $\langle location : value @ timestamp \rangle$



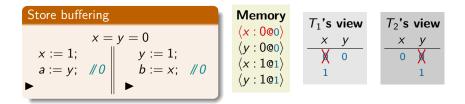
Global memory is a pool of messages of the form

 $\langle location : value @ timestamp \rangle$



Global memory is a pool of messages of the form

 $\langle location : value @ timestamp \rangle$



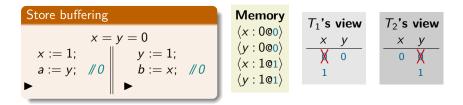
Global memory is a pool of messages of the form

 $\langle location : value @ timestamp \rangle$

Store bufferingMemoryx = y = 0 $\langle x : 0@0 \rangle$ x := 1;y := 1;a := y; #0b := x; #0 \downarrow (x : 1@1) \downarrow f_1 's viewx = yy := 1;b := x; #0 $x : 1@1 \rangle$ \downarrow f_2 's view \downarrow x = y \downarrow f_2 's view \downarrow x = y \downarrow f_2 's view \downarrow f_2 's view<

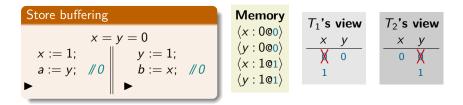
Coherence test

$$x = 0$$
 $x := 1;$
 $a := x; \ // 2$
 $x := 2;$
 $b := x; \ // 1$



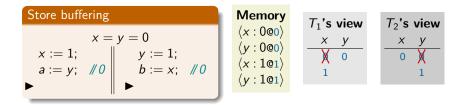
Coherence test

$$\begin{array}{c|c} x = 0 \\ & x := 1; \\ a := x; \ // 2 \end{array} \xrightarrow{k : x := 2; \\ b := x; \ // 1 \end{array} \xrightarrow{k : x := 0} \begin{array}{c|c} Memory \\ \langle x : 0 @ 0 \rangle \end{array} \xrightarrow{k} \begin{array}{c} T_1 \text{'s view} \\ x \\ \hline 0 \\ \hline 0 \end{array} \xrightarrow{k} \begin{array}{c} x \\ \hline 0 \\ \hline \end{array} \xrightarrow{k} \begin{array}{c} 0 \\ \hline 0 \\ \hline \end{array} \xrightarrow{k} \begin{array}{c} 0 \\ \hline \end{array}$$



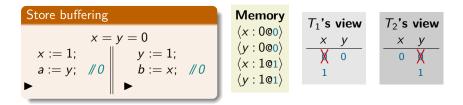
Coherence test

$$\begin{array}{c|c} x = 0 \\ x := 1; \\ \bullet a := x; \ // 2 \end{array} \quad \bullet x := 2; \\ b := x; \ // 1 \end{array} \quad \begin{array}{c|c} Memory \\ \langle x : 0@0 \rangle \\ \langle x : 1@1 \rangle \end{array} \quad \begin{array}{c} T_1 \text{'s view} \\ x \\ \hline X \\ 1 \end{array} \quad \begin{array}{c} x \\ \hline x \\ 0 \end{array}$$



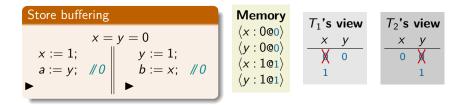
Coherence test

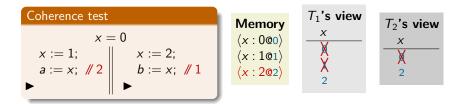
$$\begin{array}{c|c} x = 0 \\ x := 1; \\ \bullet a := x; \ // 2 \end{array} \xrightarrow{x := 2; \\ b := x; \ // 1} \begin{array}{c|c} Memory \\ \langle x : 0 @ 0 \rangle \\ \langle x : 1 @ 1 \rangle \\ \langle x : 2 @ 2 \rangle \end{array} \xrightarrow{T_1's view} \\ x \\ \hline x \\ 1 \\ \hline \end{array} \xrightarrow{x} \\ \hline x \\ 2 \end{array}$$



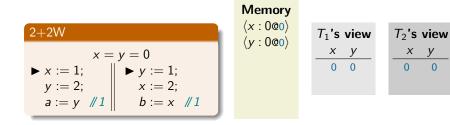
Coherence test

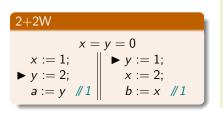
$$\begin{array}{c|c}
x = 0 \\
x := 1; \\
a := x; // 2 \\
\end{array} \xrightarrow{x := 2; // 1} \begin{array}{c|c}
Memory \\
\langle x : 0@0 \rangle \\
\langle x : 1@1 \rangle \\
\langle x : 2@2 \rangle \end{array} \xrightarrow{T_1's view \\
x \\
\hline x \\
2 \\
\end{array} \xrightarrow{x} \\
\hline x \\
2 \\
\end{array}$$

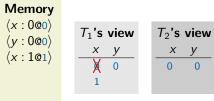


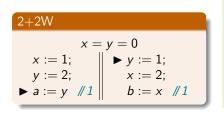


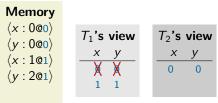
2+2W x = y = 0 x := 1; y := 1; y := 2;a := y //1 b := x //1

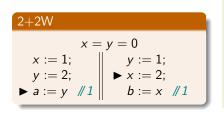








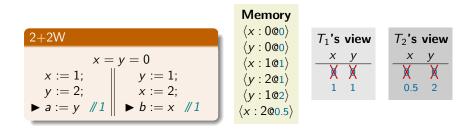




$$\begin{array}{l} \textbf{Memory} \\ \langle x: 0@0 \rangle \\ \langle y: 0@0 \rangle \\ \langle x: 1@1 \rangle \\ \langle y: 2@1 \rangle \\ \langle y: 1@2 \rangle \end{array}$$

$$\begin{array}{c|c} T_1 \text{'s view} \\ \hline x & y \\ \hline \hline x & \chi \\ 1 & 1 \end{array}$$

$$\begin{array}{c|c} T_2 \text{'s view} \\ \hline x & y \\ \hline 0 & \swarrow \\ & 2 \\ \end{array}$$



Writes choose timestamp greater than the thread's view, not necessarily the globally greatest one.

Load buffering

Load buffering (LB)

$$x = y = 0$$

$$a := x //1 || b := y //1$$

$$y := 1 || x := 1$$

- COH allows this outcome.
- But, the suggested operational semantics disallows it!
- We will see later an approach to fix this mismatch (using out-of-order execution).
- ► For now, we will strengthen the declarative semantics.

Declarative semantics for strong coherence

Definition (Strong coherence)

An execution *G* is *strongly coherent* if the following hold:

- G is complete.
- *G* is coherent wrt some modification order mo for *G*.
- ► G.po ∪ G.rf is acyclic.

A note about the implementability of StrongCOH

Some hardware implementations (*e.g.*, ARM) allow $po \cup rf$ cycles involving only plain loads and stores. To implement StrongCOH on those architectures, a syntactic dependency or a fence has to be introduced between every load and subsequent store.

- Time $\triangleq \{t \in \mathbb{Q} \mid t \ge 0\}$ is the set of timestamps.
- A message is a triple ⟨x : v@t⟩ where x ∈ Loc, v ∈ Val, and t ∈ Time.
- A memory is a finite set of messages.
- A view is a function *view* : Loc \rightarrow Time.
- A thread view function is a function V : Tid → (Loc → Time) assigning a view to every thread.

The state consists of

- ▶ a program P
- a store function S
- a memory M
- a thread view function V

Initial state $\langle P, S_0, M_0, V_0 \rangle$ where

- $\blacktriangleright S_0 = \lambda i. \ s_0 = \lambda i. \ \lambda r. \ 0$
- $\blacktriangleright \ M_0 = \{ \langle x : 0@0 \rangle \mid x \in \mathsf{Loc} \}$
- $V = \lambda i$. view₀ = λi . λx . 0.

Machine transitions

$$\begin{array}{l} \text{READ} \\ \text{SILENT-THREAD} \\ \frac{P, S \xrightarrow{i:t}}{P, S, M, V \Rightarrow P', S', M, V} \end{array} \xrightarrow{P, S \xrightarrow{i:l}} P', S' \qquad I = \mathbb{R}(x, v) \\ \langle x : v @t \rangle \in M \qquad V(i)(x) \leq t \\ view' = V(i)[x \mapsto t] \\ \hline P, S, M, V \Rightarrow P', S', M, V \end{array}$$

WRITE

$$P, S \xrightarrow{i:l} P', S' \qquad l = W(x, v)$$

$$V(i)(x) < t \qquad \forall v'. \langle x : v'@t \rangle \notin M$$

$$\frac{M' = M \cup \{\langle x : v@t \rangle\} \qquad view' = V(i)[x \mapsto t]}{P, S, M, V \Rightarrow P', S', M', V[i \mapsto view']}$$

Definition (Operational StrongCOH)

An outcome *O* is allowed for a program *P* under StrongCOH if there exist *M*, *V* such that $P, S_0, M_0, V_0 \Rightarrow^* \text{skip} \| \dots \| \text{skip}, O, M, V$.

Correspondence proof

As for SC, we will introduce an "intermediate" semantics for StrongCOH.

Establish correspondence between operational StrongCOH and declarative StrongCOH in two steps:

- 1. operational StrongCOH = intermediate StrongCOH
- 2. declarative StrongCOH = intermediate StrongCOH

Operational version of StrongCOH declarative semantics

State $\langle P, S, G, \text{mo} \rangle$ where $P \in \text{Program}$, $S \in (\text{Tid} \rightarrow \text{Store})$, $G \in \text{ExecutionGraph}$, $\text{mo} \subseteq G.E \times G.E$.

- Initial stores: $S_0 \triangleq \lambda i. s_0$
- ▶ Initial execution: G₀ consisting only of the initialization events

• Initial modification order: $mo_0 = \emptyset$

$$\begin{array}{l} \text{NON-SILENT} \\ \text{SILENT} \\ \underline{P, S \xrightarrow{i:\varepsilon} P', S'} \\ \overline{P, S, G, \text{mo} \Rightarrow P', S', G, \text{mo}} \end{array} \begin{array}{l} \text{NON-SILENT} \\ P, S \xrightarrow{i:l} P', S' & l \neq \varepsilon \\ G' \in \text{Add}(G, \langle n, i, l \rangle, i) & \text{mo} \subseteq \text{mo}' \\ \text{mo}' \text{ is a modification order for } G' \\ G' \text{ is COH-consistent wrt mo}' \\ \overline{P, S, G, \text{mo} \Rightarrow P', S', G', \text{mo}'} \end{array}$$

Definition (Operational-declarative StrongCOH)

An outcome *O* is allowed for a program *P* under StrongCOH if there exist G, mo such that $P, S_0, G_0, mo_0 \Rightarrow^* skip || ... || skip, O, G, mo.$

$Operational\ StrongCOH = intermediate\ StrongCOH$

Our bisimulation relation:

 $P, S, M, V \sim P', S', G,$ mo if the following hold:

- ► *P* = *P*′
- ► *S* = *S*′

• there exists a function $ts : G.W \rightarrow Time$ such that:

- $ts(w_1) < ts(w_2)$ whenever $\langle w_1, w_2 \rangle \in m_0$
- $M = \{ \langle \texttt{loc}(w) : \texttt{val}_w(w) @ts(w) \rangle \mid w \in G. \emptyset \}$
- ► $V = \lambda i x. \max\{ts(w) \mid w \in dom([G.W_x]; G.rf^?; [G.E^i])\}$

► *G* is strongly coherent (wrt mo).

Exercise

- Show that \sim is a bisimulation relation.
- Hence deduce that the operational StrongCOH model and the intermediate StrongCOH model have the same outcomes for any given program.

$\mathsf{Declarative\ StrongCOH}\subseteq\mathsf{intermediate\ StrongCOH}$

- ► Let *G* be an StrongCOH-consistent execution graph of *P*₀ with outcome *O*.
- Let mo be a modification order for G such that G is COH-consistent wrt mo.
- ▶ We show that $P_0, S_0, G_0, \mathsf{mo}_0 \Rightarrow^* \mathsf{skip} \| ... \| \mathsf{skip}, O, G, \mathsf{mo}.$
- ▶ Let $a_1, ..., a_n$ be an enumeration of $G.E \setminus E_0$ following $G.po \cup G.rf$.
- ▶ By the "execution generation" lemma, there exist $\langle P_1, S_1 \rangle, \dots, \langle P_n, S_n \rangle$ such that:

•
$$P_n = \operatorname{skip} \| \dots \| \operatorname{skip} \text{ and } S_n = O$$

• For every $1 \le j \le n$, we have:

$$P_{j-1}, S_{j-1} \xrightarrow{\texttt{tid}(a_j):\varepsilon} {}^* \xrightarrow{\texttt{tid}(a_j):\texttt{lab}(a_j)} \xrightarrow{\texttt{tid}(a_j):\varepsilon} {}^* P_j, S_j$$

• For every $0 \le j \le n$, let

- G_j the restriction of G to $E_0 \cup \{a_1, \dots, a_j\}$
- ▶ mo_j the restriction of mo to $E_0 \cup \{a_1, ..., a_j\}$

▶ Then, for every $1 \le j \le n$, we have: (why?)

$$P_{j-1}, S_{j-1}, G_{j-1}, \operatorname{mo}_{j-1} \Rightarrow^* P_j, S_j, G_j, \operatorname{mo}_j$$

$\mathsf{Operational}\text{-}\mathsf{declarative}\ \mathsf{StrongCOH}\subseteq\mathsf{declarative}\ \mathsf{StrongCOH}$

- ▶ Suppose that $P_0, S_0, G_0, \mathsf{mo}_0 \Rightarrow^* \mathsf{skip} \| ... \| \mathsf{skip}, O, G, \mathsf{mo}.$
- ▶ We show that *G* is a StrongCOH-consistent execution graph of *P* with outcome *O*.
- We know:

$$P_0, S_0, G_0, \mathsf{mo}_0 \Rightarrow P_1, S_1, G_1, \mathsf{mo}_1 \Rightarrow ... \Rightarrow P_n, S_n, G_n, \mathsf{mo}_n$$

where $P_n, S_n, G_n, \mathbf{mo}_n = \mathbf{skip} \| \dots \| \mathbf{skip}, O, G, \mathbf{mo}.$

- ▶ By definition, *G* is COH-consistent.
- ► Using induction on the length of the sequence, we also have that G.po ∪ G.rf is acyclic.
- It remains to show that each Gⁱ is an execution of P₀(i) with final store O(i). (This is done exactly as for SC.)

Operational semantics for RA

Release/acquire synchronization

Can we extend the operational semantics to support message passing (*i.e.*, release-acquire synchronization)?

Message passing (MP) x = y = 0 x := 42; || a := y; // 1y := 1 || b := x // 0

Double message passing

$$x = y = 0$$
 $x := 42;$
 $a := y; // 1$
 $b := z; // 1$
 $y := 1$
 $z := 1$
 $c := x // 0$

Desired semantics

When reading a message the thread becomes aware of all messages that the writer of the message was aware of when the message was written.

We implement this using *message views*:

- Each message m will carry a view: the view of the thread who wrote m when m was written.
- ▶ When reading a message *m*, the thread will update its view to include at least the view contained in *m*.

Operational semantics for RA

- A message is a tuple ⟨x : v@t view⟩ where x ∈ Loc, v ∈ Val, t ∈ Time and view : Loc → Time
- ▶ Initially, $M_0 \triangleq \{ \langle x : 0@0 \perp \rangle \mid x \in Loc \}$
- Bottom view: $\bot \triangleq \lambda x$. 0
- ▶ Joining views: $view_1 \sqcup view_2 \triangleq \lambda x$. max{ $view_1(x)$, $view_2(x)$ }

$$\begin{array}{l} \text{READ} \\ P, S \xrightarrow{i:l} P', S' \qquad l = \mathbb{R}(x, v) \\ \hline P, S, M, V \Rightarrow P', S', M, V \end{array} \end{array}$$

WRITE

$$\begin{array}{ccc} P, S \xrightarrow{i:l} P', S' & l = \mathbb{W}(x, v) \\ V(i)(x) < t & \forall v', view. \ \langle x : v'@t & view \rangle \notin M \\ \hline view' = V(i)[x \mapsto t] & M' = M \cup \{ \langle x : v@t & view' \rangle \} \\ \hline P, S, M, V \Rightarrow P', S', M', V[i \mapsto view'] \end{array}$$

Exercise: RA operational semantics

Definition (Operational RA)

An outcome *O* is allowed for a program *P* under RA if there exist *M*, *V* such that *P*, S_0 , M_0 , $V_0 \Rightarrow^* \mathbf{skip} \parallel ... \parallel \mathbf{skip}$, *O*, *M*, *V*.

Exercise

Prove the correspondence between the declarative and the operational definitions of RA.

Exercise: Strong release/acquire semantics

Suppose we change the write step in the operational semantics of RA as follows:

$$\begin{array}{ccc} & P, S \xrightarrow{i:l} P', S' & I = \mathbb{W}(x, v) \\ & \forall t', v', view. \langle x : v'@t' & view \rangle \in M \Rightarrow t' < t \\ & \underline{view' = V(i)[x \mapsto t]} & M' = M \cup \{ \langle x : v@t & view' \rangle \} \\ \hline & P, S, M, V \Rightarrow P', S', M', V[i \mapsto view'] \end{array}$$

Here, when writing a message, the thread may only choose a timestamp larger than *all* timestamps that were used for the given location.

- Show an example which differentiates this model from RA.
- What will be the corresponding declarative semantics?

Exercise: Plain accesses in Java

Recall the following alternative definition of coherence:

Let mo be a modification order for an execution graph G. G is coherent wrt mo iff the following hold:

- rf; po is irreflexive.
- mo; po is irreflexive.
- mo; rf; po is irreflexive.
- ▶ rf⁻¹; mo; po is irreflexive.
- rf⁻¹; mo; rf; po is irreflexive.

(no-future-read)
(coherence-ww)
(coherence-rw)
(coherence-wr)
(coherence-rr)

Plain accesses in the Java memory model do not provide full coherence. In particular, they do not ensure "coherence-rr".

 Adapt the StrongCOH timestamp machine to match this weaker variant.

Further reading

- Taming release-acquire consistency. Ori Lahav, Nick Giannarakis, Viktor Vafeiadis. POPL 2016: 649-662
- A promising semantics for relaxed-memory concurrency. Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, Derek Dreyer. POPL 2017: 175-189